COMP1409: Introduction to Software Development I

Lesson 2

Agenda

- Quiz
  - Quiz 1
  - Review Answers
- Logistics
- Designing Classes
- Lesson 2
  - Comments
  - Constructors
  - Basic Java Syntax
- Lab 2
  - Part A – Due In Class
  - Part B – Due Wednesday at Midnight on D2L

Quiz 1

- 15 minutes
- Put up your hand when you are done, I will collect your paper
- We will reviews the answers afterwards
- Your marks will be in D2L this afternoon

Logistics

- Lab 1 – Marks recorded on D2L. Please make sure yours is there.
- Assignment 1 – Will be posted to D2L after next class.
  - Due Week 7 (Oct. 20th)
- Online Java Books – Free Safari Books Access
  - https://ezw.lib.bcit.ca/login?url=http://proquest.safaribooksonline.com
  - Login with your BCIT Student Number and Password
  - Examples:
    - *Beginners Java* - http://proquestcombo.safaribooksonline.com.ezw.lib.bcit.ca:2048/book/programming/java/9780992133047
    - *Think Java* – http://proquestcombo.safaribooksonline.com.ezw.lib.bcit.ca:2048/book/programming/java/9781491929551
  - BlueJ - https://www.bluej.org/help/javahelp.html
    - Java resources from the BlueJ team

Designing Classes

- Need to know the context of the software in which your class will be used
- For example:

Car

Used for inventory of cars on a used car lot

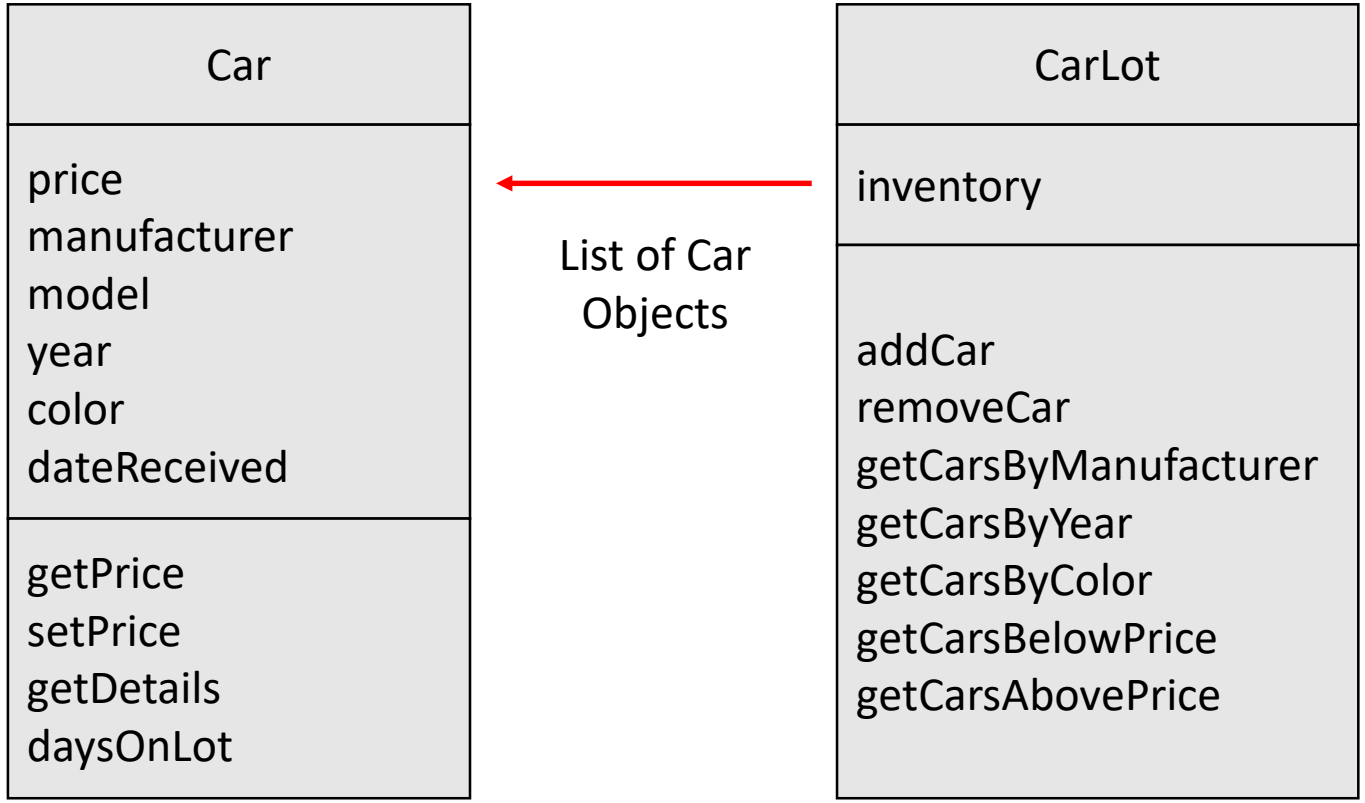
Car

Used for the state and location of the car in a production line

Car

Used to store the current sensor values for a car in an engine management system

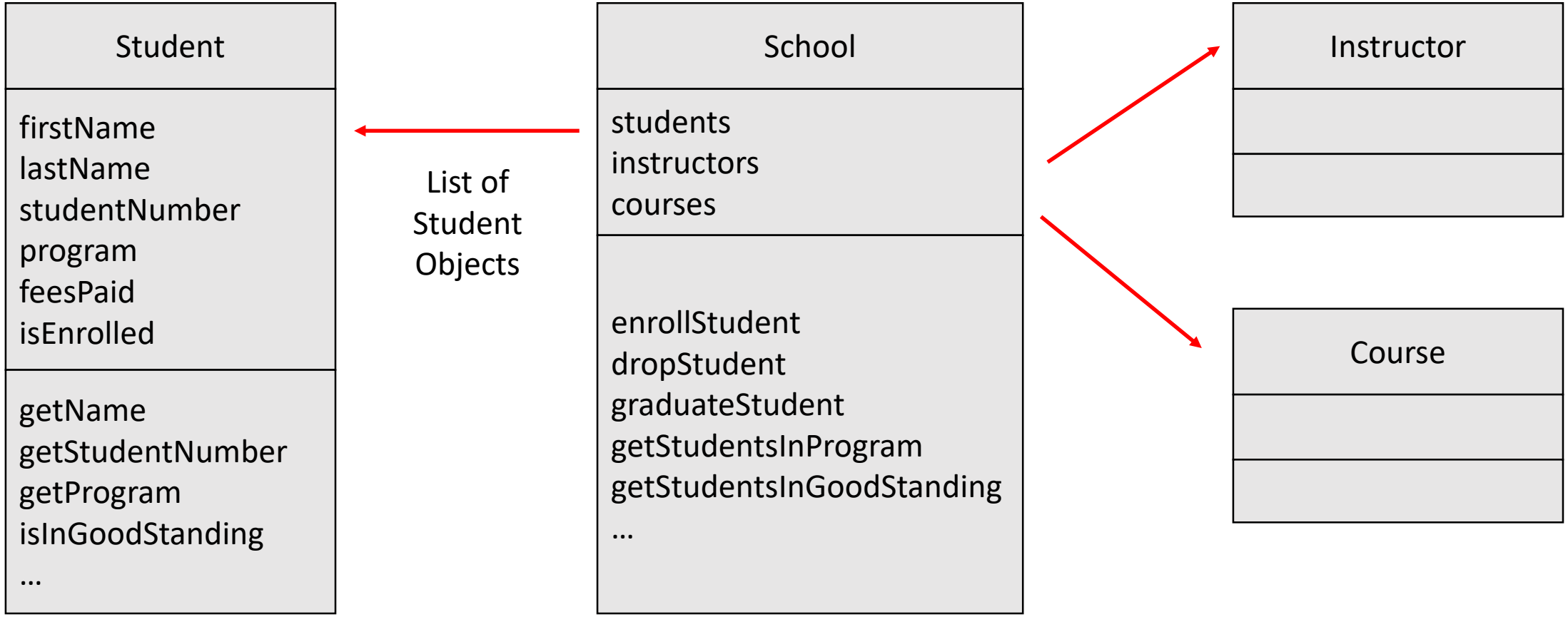
Designing Classes – Example (Car)



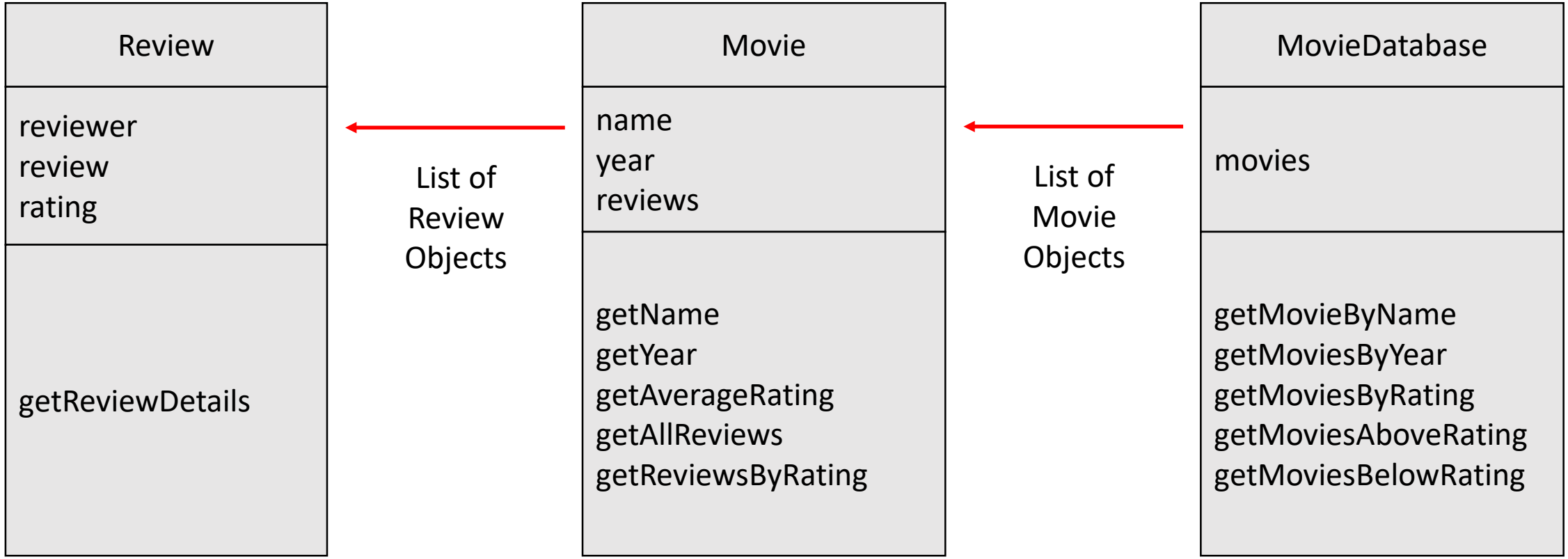
This type of relationship in Object Oriented Programming is called **Composition** – one object contains one or more instance of another object.

We will be building these class relationships later in the course.

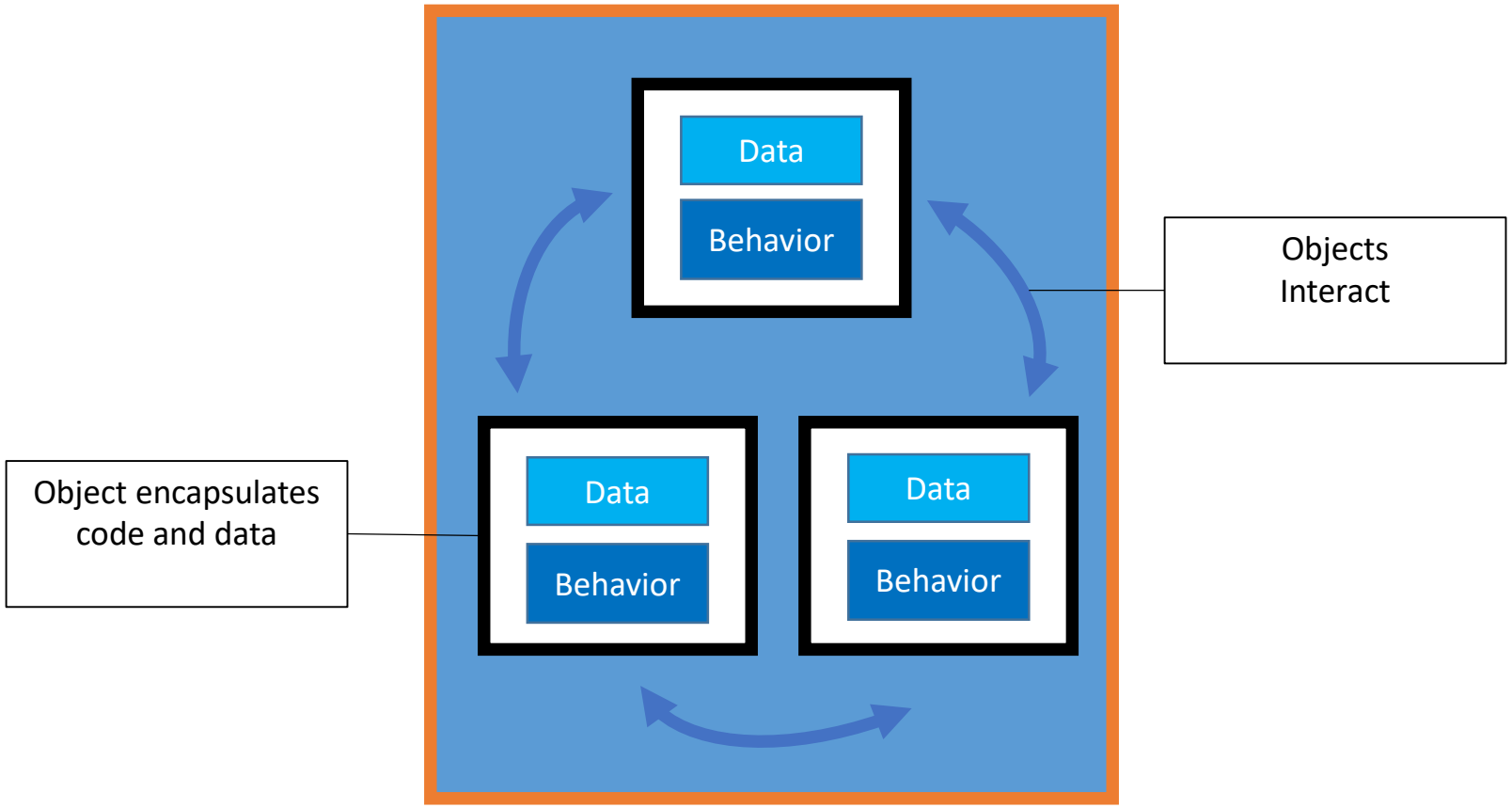
Designing Classes – Another Example (Student)



Designing Classes – Last Example (Movie)



Object Oriented Programming

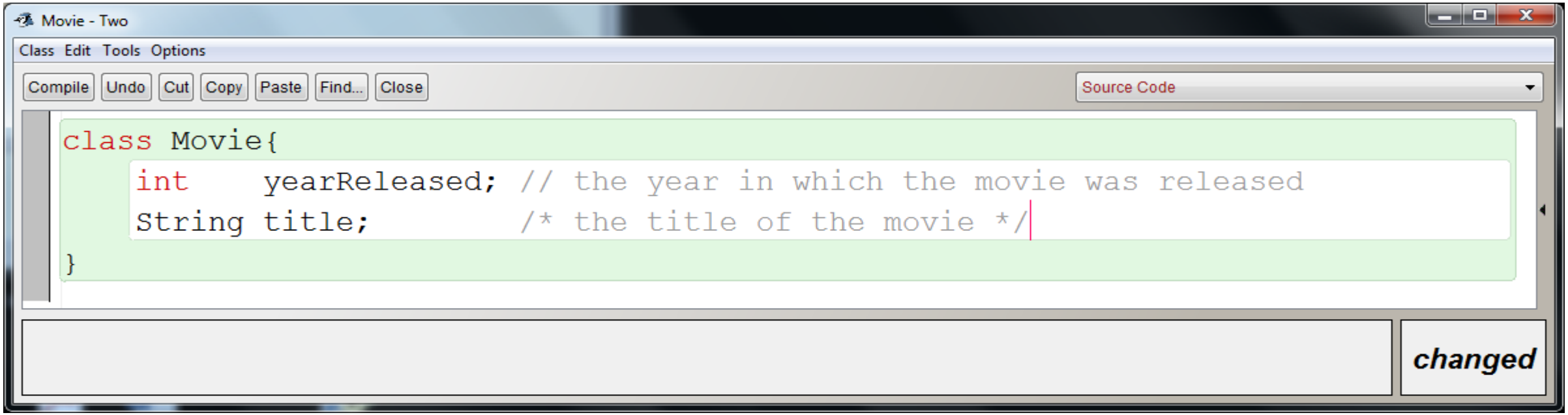


Learning Outcomes: Lesson 2

- Comments
- Visibility modifiers (“access modifiers”)
- if statements
- null
- Constructors
- Parameters
- Assignment statement =
- Logical operators
- IllegalArgumentException
- Relational operators > >= < <= == !=

Comments

- Comments are ignored by the Java compiler
- They are used as “notes” for developers
- There are two types of comments in Java, and three “styles” for using them.



Comments – Types and Styles

Single Line

```
// This is a comment

/* This is a comment */
```

Used to comment a single line

Multi-Line

```
/* Line 1 of the comment
   Line 2 of the comment
*/

/** Line 1 of the comment
 *  Line 2 of the comment
 */
```

Used to comment multiple lines of code

JavaDoc

Multi-line comment with a specific content

Note that JavaDoc uses this style:

```
/**
 * @author Bill Smith
 * @version 1.0
 */
```

Used to create documentation for your class

Comments - Examples

Movie - Two

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close Source Code

```
class Movie{
    int     yearReleased; // the year in which the movie was released
    String title;         /* the title of the movie */
}
```

changed

JavaDoc Comments

Movie - Two

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close Source Code

```
/**
 * @author Tiger Woods
 * @version 1.0
 * This class defines a Movie.
 */
public class Movie{
    int    yearReleased; // the year in which the movie was released
    String title;        /* the title of the movie */
}
```

Class compiled - no syntax errors

saved

JavaDoc Comments

Movie - Two

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close Source Code

```
/**
 * @author Tiger Woods
 * @version 1.0
 * This class defines a Movie.
 */
public class Movie{
    int    yearReleased; // the year in which the movie was released
    String title;        /* the title of the movie */
}
```

Class compiled - no syntax errors

saved

Movie - Two

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close Source Code

Source Code

Documentation

```
/**
```

Movie - Two

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close Documentation

Class Movie

```
java.lang.Object
  └Movie
```

```
public class Movieextends java.lang.Object
```

Version:

1.0 This class defines a Movie.

Author:

Tiger Woods

Field Summary

(package private) java.lang.String	title
(package private) int	yearReleased

Loading class interface... Done.

saved

Comments – Best Practices

- Except for JavaDoc, use comments to describe the *why* (or to clarify) rather than the *what*
  - **Bad:** int modelYear; // model year
  - **Bad:** printCarDetails(); // Prints the details of the car
- Don’t overuse comments
  - Use good clear naming in your code
  - Don’t comment every line
  - Don’t write essays in your comments - no one will read them anyways!
- Block comments are generally preferred over inline comments

Visibility modifiers

- Visibility modifiers are also called access modifiers
- Instance variables, methods, and constructors can be declared as public or private (there are also other modifiers, to be discussed later)
- We will always make our instance variables private
- We can make our classes and constructors public
- We will make some methods public, others private
- private means "accessible only from within this class"
- public means "accessible from any class".

Visibility modifiers

Movie - Two

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close Source Code

```
/**
 * @author Tiger Woods
 * @version 1.0
 * This class defines a Movie.
 */
public class Movie{
    private int    yearReleased; // the year in which the movie was released
    private String title;        /* the title of the movie */
}
```

changed

Constructors

- A constructor in a class has the exact same name as the class
- It is called automatically when an object is created
- Its job is to ensure that the instance variables are set to proper/normal/valid/sensible initial values (remember, otherwise Java will give its default values)
- A constructor should not normally be marked private.

Constructors

- Remember, the default value for an int is 0 and for a String is null
- We don't want to allow the creation of a Movie object which was released before 1895, nor of a movie whose title is null.

Movie - Two

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close Source Code

```java
/**
 * @author Tiger Woods
 * @version 1.0
 * This class defines a Movie.
 */
public class Movie{
    private int    yearReleased; // the year in which the movie was released
    private String title;        /* the title of the movie */

    public Movie(int theYearReleased, String theTitle){
        if(theYearReleased >= 1895){
            yearReleased = theYearReleased;
        }

        if(theTitle != null){
            title = theTitle;
        }
    }
}
```

changed

Constructors

- Since this constructor is public it should have a Javadoc comment too, with @param tags for each parameter

Movie - Two

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close Source Code

```
/**
 * @author Tiger Woods
 * @version 1.0
 * This class defines a Movie.
 */
public class Movie{
    private int    yearReleased; // the year in which the movie was released
    private String title;        /* the title of the movie */

    public Movie(int theYearReleased, String theTitle){
        if(theYearReleased >= 1895){
            yearReleased = theYearReleased;
        }

        if(theTitle != null){
            title = theTitle;
        }
    }
}
```

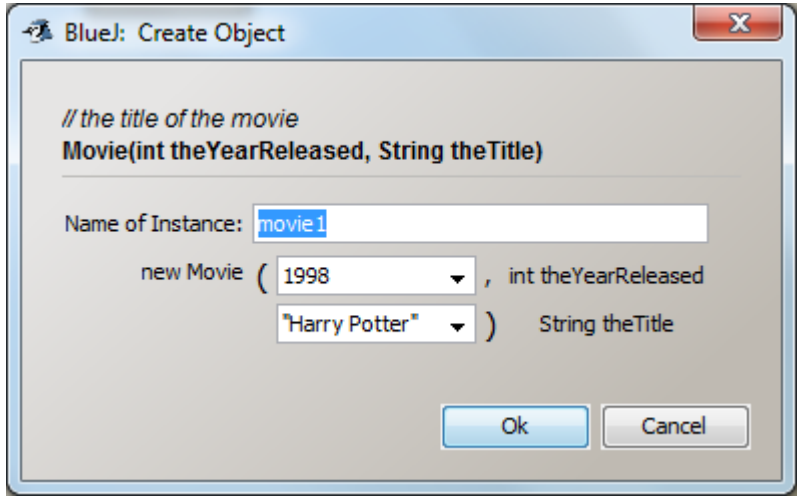
changed

Constructor Parameters

- The Movie constructor takes two parameters.

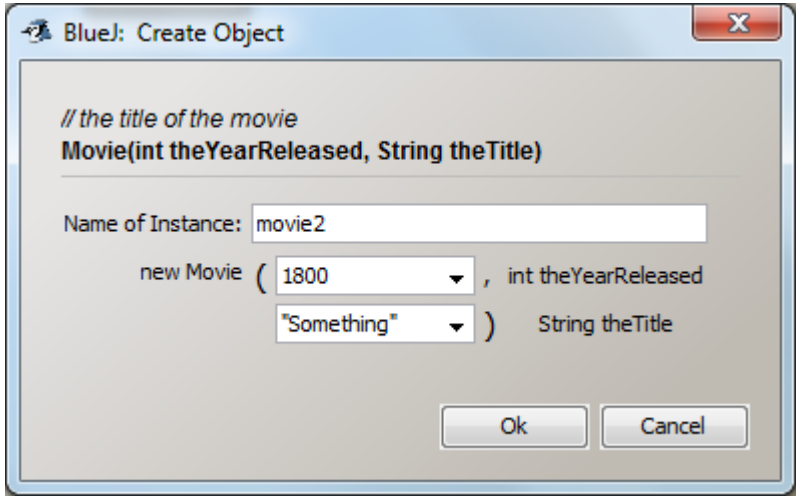
  public Movie(**int theYearReleased**, **String theTitle**)

- theYearReleased and theTitle are also known as “formal arguments”
- “Actual” arguments would be something like 1998 and “Harry Potter”, for example:



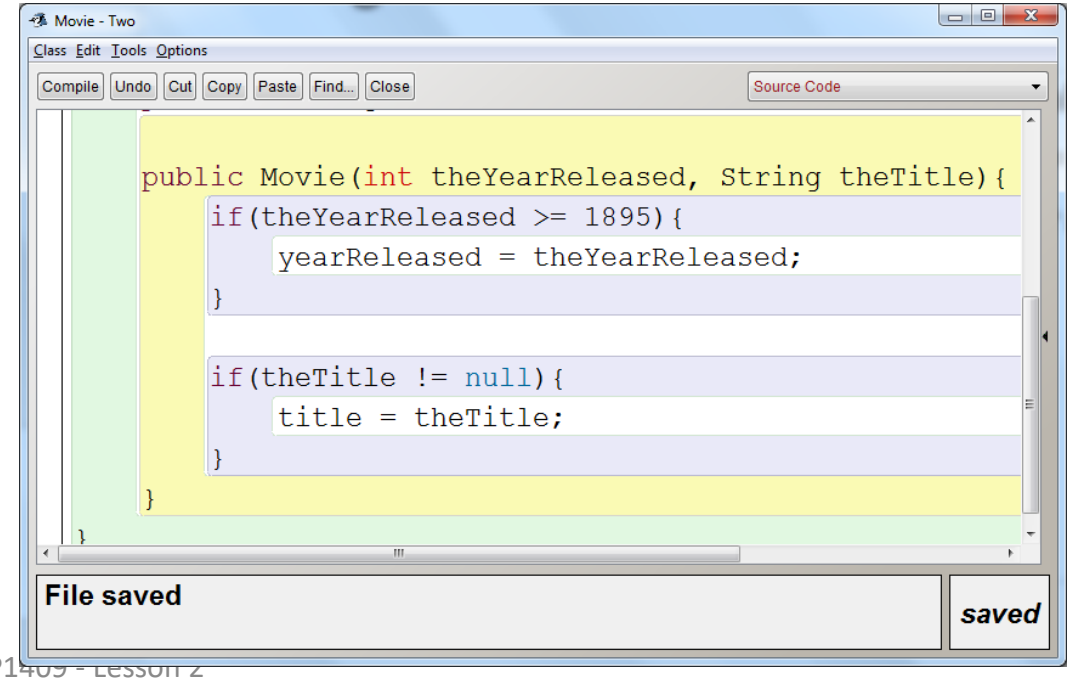
Constructor Parameters

- The constructor did its job.
- The “bad” data of 1800 was rejected.



Assignment statement

- The equals sign = by itself does not mean equals
- To say “equals” in Java, use two equals signs ==
- One equals sign alone *assigns the value on the right side, to the variable on the left side*.



if statements

- One way for a Java program to make decisions is to use “if statements”.
- Example:

Person - two

Class Edit Tools Options

City Person

Compile Undo Cut Copy Paste Find... Close Source Code

```
    private String middleName;
    private String countryOfBirth;

    Person(double weightInKilograms, int yearOfBirth)
    {
        if(weightInKilograms <= 635)
        {
            kg = weightInKilograms;
        }

        if(yearOfBirth <= 2018)
        {
            yearBorn = yearOfBirth;
        }
    }
}
```

saved

if statements

- <u>if</u> statements are very often accompanied by an (optional) <u>else</u> statement
- In between these, there may be zero, one, or many <u>else if</u> statements also.

Car - two

Class Edit Tools Options

Car | NHLHockeyPlayer | Person | City

Compile Undo Cut Copy Paste Find... Close Source Code

```
        }

        if(theMake.equals("dodge") || theMake.equals("chevrolet") || theMake.equals("ford")){
            make     = theMake;
            country  = "america";
        }else if(theMake.equals("honda") || theMake.equals("nissan")){
            make    = theMake;
            country = "japan";
        }else if(theMake.equals("porsche") || theMake.equals("mercedes")){
            make    = theMake;
            country = "germany";
        }else{
            throw new IllegalArgumentException("invalid make");
        }
    }
}
```

saved

null

- null means “no object” (or “no reference”…more on that later) for object / reference data types
- Not relevant for primitive data types.

IllegalArgumentException

Ends the code (e.g. ends the constructor) immediately.

NHLHockeyPlayer - two

Class Edit Tools Options

Car NHLHockeyPlayer Person City

Compile Undo Cut Copy Paste Find... Close Source Code

```
class NHLHockeyPlayer{
    private String teamName;
    private int    number;
    private String lastName;

    NHLHockeyPlayer(String hockeyTeamName, int jerseyNumber, String surName){
        if(hockeyTeamName != null){
            teamName = hockeyTeamName;
        }else{
            throw new IllegalArgumentException("name is not supposed to be null");
        }
```

saved

Logical operators

- There are three logical operators: and, or, not.
- In general these are evaluated left to right, when they are put in combination with one another.
- and is written as &&
  today is Saturday && today is Sunday && ~~today is Monday~~
- or is written as ||
  today is Friday || today is Saturday || ~~today is Sunday~~
- not is written as !

  boolean x = true; x = !x; // x is now false
- Note: not equals is written as !=

Relational operators

- == Equals
- != Not Equals
- >= Greater Than or Equals To
- <= Less Than of Equals To
- > Greater Than
- < Less Than

Note: do not use with Strings or other objects; use .equals()

Note: do not use with Strings or other objects; use !.equals()

Examples – Logical and Relational Operators

```
int x = 1;
int y = 2;
int z = 3;

if ((x < y) && (y > z)) {
  // true or false?
  // false: y is not greater than z and both must be true for &&
}

if ((x <= y) || (y >= z)) {
  // true or false?
  // true: x is less than y and only one must be true for ||
}

if (!(x == y) && (y != z)) {
  // true or false?
  // true: x doesn’t equal y is false, but the ! makes it true, y is not equal to z
}
```

Questions…

- What are the two types and three styles of Comments?
- What is a Constructor? How do we pass data to a Constructor?
- How do we assign values to variables?
- What do we use to make decisions in our code?
- What is an example of a logical operator? A relational operator?

Lab 2

- Part A – In-class
  - Must be signed off before you leave
- Part B – In-class/take-home
  - Must be uploaded to D2L by Wednesday (Sept. 19 at midnight)

Remember: You can't save files to the lab computers. Either e-mail to yourself or use ShareFile (https://kb.bcit.ca/sr/sharefile/2052.shtml)